NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

BOKMÅL

KONTINUASJONSEKSAMEN I FAG
TDT4100 Objekt-orientert programmering

Onsdag 17. august 2005
KL. 09.00 – 13.00

Faglig kontakt under eksamen:
Hallvard Trætteberg, tlf (735)93443 / 918 97263
Trond Aalberg, tlf (735)97952 / 976 31088

Tillatte hjelpemidler:

- Lewis & Loftus: Java Software Solutions (alle utgaver)
- Winder & Roberts: Developing Java Software
- Lervik & Havdal: Programmering i Java
- Mughal, Hamre & Rasmussen: Java som første programmeringsspråk
- Arnold, Gosling & Holms: The Java Programming Language
- Lervik & Havdal: Java the UML Way
- Liang: Introduction to Java programming
- Lewis & Loftus: Java Software Solutions
- Horton: Beginning Java 2 SDK 1.4 Edition
- Brunland,Lingjærde & Maus: Rett på Java
- Lemay/Cadenhead: SAMS Teach Yourself Java 2 platform in 21 days.

Sensurdato:
7. sept. 2005. Resultater gjøres kjent på http://studweb.ntnu.no/ og sensurtelefon 81 54 80 14.

Prosentsatser viser hvor mye hver oppgave teller innen settet.

Merk: All programmering skal foregå i Java.

Lykke til!

OPPGAVE 1 (10%): Iterasjon i tabeller og List.

a) Skriv kode for en metode `Object[] reverser1(Object[] tabell)` som returnerer en kopi av `tabell` med alle elementene i *motsatt* rekkefølge. Mao. skal første element i `tabell` være sist i resultattabellen, element nr. 2 i `tabell` skal være nest sist, osv.

Løsningselementer:

- lage kopi med riktig type og lengde
- iterere over alle elementene (i originaltabellen eller kopien)
- putte riktig element fra original-tabellen på riktig plass i kopien

```
Object[] reverser1(Object[] tabell) {
      Object[] resultat = new Object[tabell.length];
      for (int i = 0; i < tabell.length; i++) {
            resultat[i] = tabell[tabell.length - i - 1];
      }
      return resultat;
   }
```

b) Skriv kode for en metode `void reverser2(List liste)` som reverserer `liste`, slik at rekkefølgen snus. Mao. det elementet i `liste` som var først før et kall til `reverser2` skal være sist etter kallet, det elementet i `liste` som var nr. 2 skal være neste sist osv. To kall etter hverandre vil gjenopprette den originale rekkefølgen. Relevante metoder i `List`-grensesnittet (som `ArrayList` implementerer):

```
// Appends the specified element to the end of this list.
boolean add(Object element)

// Inserts the specified element at the specified position.
void add(int index, Object element)

// Removes the element at the specified position in this list.
// Returns the removed element.
Object remove(int index)

// Removes the first occurrence in this list
// of the specified element.
Boolean remove(Object element)

// Returns the element at the specified position in this list.
Object get(int index)

// Replaces the element at the specified position in this list
// with the specified element.
Object set(int index, Object element)

// Returns the number of elements in this list.
int size()
```

Her det viktig å merke seg at det ikke skal lages noen kopi, men endre lista som gis som parameter. Det er mulig å gå via en kopi, men dette trekkes det for. To forslag vises under, en som kun bruker get og set og en som kun brukes remove og add. For den første er det vesentlig at en bare går gjennom halve lista og bytter om elementene riktig. For den andre er det vesentlig at en går baklengs og legger til sist (alternativt går forlengs og legger til først).

```
public void reverser2(List elementer) {
```

```
      for (int i = 0; i < elementer.size() / 2; i++) {
            Object o1 = elementer.get(i);
            Object o2 = elementer.get(elementer.size() - i - 1);
            elementer.set(i, o2);
            elementer.set(elementer.size() - i - 1, o1);
      }
   }

   public void reverser2(List elementer) {
      for (int i = elementer.size() - 1; i >= 0; i--) {
            Object o = elementer.remove(i);
            elementer.add(o);
      }
   }
```

OPPGAVE 2 (15%): Klasse og Iterator.

a) Skriv kode for en Teller-klasse, som har et teller-attributt av typen int og en tell-metode. Tell-metoden skal øke teller-attributtet med 1 hver gang den kalles.

b) Utvid Teller-klassen med relevante attributter og konstruktør(er), slik at teller-attributtet *starter på* en bestemt start-verdi og *stopper på* en bestemt slutt-verdi (dvs. når slutt-verdien, men telles deretter ikke videre).

a)-oppgaven er planke. Merk at vi ikke ber om innkapsling, så bruk av synlighetsmodifikatorer er ikke viktig. En trenger ikke huske start-verdien, når teller settes til den likevel. Testen i tell-klassen er viktig å få rett, slik at stopp-verdien nås, men ikke passeres.

```
class Teller implements Iterator {
   int teller;
   int slutt;
   public Teller(int start, int slutt) {
      teller = start;
      this.slutt = slutt;
   }

   public void tell() {
      if (teller < slutt) {
            teller += 1; // evt. teller = teller + 1 eller teller++;
      }
   }
}
```

c) Skriv kode for nødvendig metoder, slik at Teller-klassen implementerer Iterator og returnerer tallene *fra og med* startverdien og *opp til, men ikke inkludert* slutt-verdien, én etter én for hvert kall til next-metoden. Kall eksisterende metoder, der du kan. Metodene i Iterator-grensesnittet er som følger:

```
// Returns true if the iteration has more elements.
boolean hasNext()

// Returns the next element in the iteration.
Object next()
```

d) Utvid implementasjonen av next-metoden slik at den kaster et unntak av typen NoSuchElementException, dersom den kalles etter at slutt-verdien er nådd.

Grenseverdien for next ift. tell er satt slik at next kan kalle hasNext og tell uten noe mer logikk. hasNext skal ikke endre på noe, kun teste. Første kall til next skal returnere start-verdien (først verdien som teller settes til), derfor må inneværende verdi tas vare på før tell kalles, for så å bli returnert. Det trekkes hvis ikke hasNext- og tell-metodene kalles fra next-metoden, og en istedenfor dupliserer deres logikk.

```
public boolean hasNext() {
   // hasNext-metoden skal ikke endre noen attributter,
   // men kun si om vi ikke ennå har nådd forbi slutt-verdien
   return teller < slutt;
}
```

```
public Object next() {
   if (! hasNext()) {
         throw new NoSuchElementException();
   }
   // Første kall til next skal returnere start-verdien.
   // Derfor må en huske nåværende teller-verdi og så kalle tell().
   Integer neste = teller; // Lager implisitt et Integer-objekt.
   tell();
   return neste;
}
```

OPPGAVE 3 (45%): Klasser og Collection-klasser.

I denne oppgaven skal du implementere en enkel kontaktbok med telefonnumre i.

a) Forklar prinsippet bak innkapsling og bruken av nøkkelordene `private` og `public`.

Innkapsling betyr at en klasses attributter skjules for andre klasser vha. `private`-nøkkelordet og at attributtene kun implisitt kan endres gjennom (innkapslings)metoder som har `public` synlighet. Dette er essensielt for at en klasse skal kunne sikre konsistent intern tilstand og ha frihet til å endre implementasjonsdetaljer uten at andre klasser må endres tilsvarende.

b) Skriv kode for en klasse Nummerliste, som skal brukes for å holde et sett telefonnumre, av typen String. Klassen skal ha et attributt kalt `numre` av typen ArrayList deklarert som `private` og en konstruktør som initialiserer `numre`-attributtet. Implementer følgende innkapslingsmetoder:
   - getAntallNumre skal returnere antall numre
   - getNummer skal returnere et gitt nummer
   - setNummer skal endre et gitt nummer
   - addNummer skal legge til et nytt nummer

Her er poenget dels å velge riktig metodesignaturer, dels å implementere dem riktig. En evt. konstruktør bør ikke ha noen parametre, ihvertfall ikke en ArrayList som brukes direkte som initialverdien til numre! Det er naturlig å gi getNummer og setNummer et indeks-parameter av typen int. I forslaget nedenfor har vi valgt å ikke gi addNummer et tilsvarende indeks-parameter, men det er greit om en har det med. Både setNummer og addNummer må selvsagt ha et String-parameter for nummeret. Returtypene må selvsagt også være riktige.

```
public class Nummerliste {

   private ArrayList numre;

   public Nummerliste() {
      numre = new ArrayList();
   }

   public int getAntallNumre() {
      return numre.size();
   }

   public String getNummer(int indeks) {
      return (String)numre.get(indeks);
   }

   public void setNummer(int indeks, String nummer) {
      numre.set(indeks, nummer);
   }

   public void addNummer(String nummer) {
      numre.add(nummer);
   }
}
```

c) I en alternativ implementasjon av Nummerliste kunne vi *arvet fra* ArrayList, istedenfor å ha et ArrayList-attributt. Forklar hvordan metodene du hittil har implementert i tilfelle måtte endres. Hva er ulempen med å arve fra ArrayList på denne måten?

Ved arv til alle `public`- og `protected`-metoder i en klasse være synlig for subklasser. I Nummerlisteklassen vil alle kall til metoder på numre-attributtet erstattes med kall til superklassens (altså ArrayList) sin metode (med eller uten super foran). I praksis betyr dette at konstruktøren blir tom og kan fjernes og at kallene til numre.size, numre.get, numre.set og numre.add erstattes med kall til hhv. size, get, set og add. Ulempen er at disse metodene er public, slik at andre klasser også kan kalle disse metodene og dermed omgå innkapslingen vår. I tillegg vil et Nummerliste-objekt være instanceof ArrayList, noe som ikke er naturlig.

d) Det er vanlig i kontaktbøker at numre kategoriseres, som f.eks. ”hjemme”, ”mobil”, ”jobb” osv. Du skal skrive kode for en subklasse av Nummerliste kalt KategorisertNummerliste, som deklarerer et nytt attributt ved navn `kategorier` av typen ArrayList. Elementene i katetorier skal tilsvare dem i numre, slik at hvert nummer i numre-lista har sin kategori i samme posisjon i kategorier-lista.

   1) Skriv kode for setNummer- og addNummer-metoder tilsvarende dem i deloppgave b), som tar inn et ekstra kategoriparameter.
   2) Redefiner den eksisterende addNummer-metoden, altså den uten katagoriparameter, slik at den setter kategorien til null.
   3) Skriv kode for en finnKategori-metode, som tar inn et nummer (en String) og returnerer tilsvarende kategori i kategorier-lista eller null hvis nummeret ikke finnes numre-lista.

Her er det viktig å skjønne sammenhengen mellom de nye metodene og de eksisterende og kunne bruke de eksisterende riktig. setNummer- og addNummer-metodene bruker begge eksisterende (hhv.) setNummer- og addNummer-metoder uten kategoriparameter, slik at logikken ikke dupliseres. Merk også hvordan redefineringen av addNummer kaller den nye addNummer med et null kategoriparameter. I alle disse tilfellene trekkes det dersom en dupliserer logikk istedenfor å kalle andre metoder.

```
public class KategorisertNummerliste extends Nummerliste {

   private ArrayList kategorier;

   public KategorisertNummerliste() {
      super();
      kategorier = new ArrayList();
   }

   public String finnKategori(String nummer) {
      for (int i = 0; i < getAntallNumre(); i++) {
            if (nummer.equals(getNummer(i))) {
                  return (String)kategorier.get(i);
            }
      }
      return null;
   }

   public void setNummer(int index, String nummer, String kategori) {
      super.setNummer(index, nummer); // evt. numre.set(...);
      kategorier.set(index, kategori);
   }

   public void addNummer(String nummer, String kategori) {
      super.addNummer(nummer) ; // evt. numre.add(...);
      kategorier.add(kategori);
   }

   public void addNummer(String nummer) {
      addNummer(nummer, null);
   }
}
```

OPPGAVE 4 (15%): Testing

a) Beskriv en generell testemetodikk som lar en teste *oppførselen til* Teller-klassen fra oppgave 2, uten at en nødvendigvis har skrevet koden selv eller har den tilgjengelig. Formuler 2 relevante regler for oppførselen til Teller-klassen.

Prinsippet er å opprette og initialisere objekter av Teller-klassen, og vekselvis kalle endringsmetoder og sjekke at lesemetodene returnerer forventede verdier, basert på reglene for Teller-klassen sin oppførsel. To relevante regler er at 1) teller-verdien etter opprettelse av Teller-objektet skal være lik startverdien oppgitt i new/konstruktør-kallet og 2) at verdien til teller-verdien skal være én større etter at tell-metoden er kalt, men kun dersom slutt-verdien ikke allerede er nådd.

b) Skriv kode for en eller flere JUnit-testmetoder som tester Teller-klassen fra oppgave 2, både metodene skrevet frem til og med deloppgave 2b) og de to skrevet i deloppgave

2c). Bruk metodene `assertEquals(Object, Object)` og `assertTrue(boolean)` i TestCase for å sjekke relevante verdier.

Her er det viktig å skrive testkode som tilsvarer reglene for oppførsel og at en tester flere kall til tell. Merk at siden vi ikke har bedt om innkapsling av teller-attributtet, er det greit om det lese direkte og ikke med f.eks. en getTeller-metode. I testTeller-metoden, tester det første kallet til assertEquals regel 1 fra spørsmål a), mens de andre kallene til assertEquals tester den andre regelen. I testIterator-metoden har både hasNext og next rollen som lesemetoder hvis returverdier testes med assertEquals/assertTrue, mens next også har rollen som endringsmetode.

```
public void testTeller() {
   Teller teller = new Teller(0, 3);
   assertEquals(teller.getTeller(), 0);
   teller.tell();
   assertEquals(teller.getTeller(), 1);
   teller.tell();
   assertEquals(teller.getTeller(), 2);
   teller.tell();
   assertEquals(teller.getTeller(), 3);
   teller.tell();
   assertEquals(teller.getTeller(), 3);
}
```

```
public void testIterator() {
   Teller teller = new Teller(0, 3);
   assertTrue(teller.hasNext());
   assertEquals(teller.next(), 0);
   assertTrue(teller.hasNext());
   assertEquals(teller.next(), 1);
   assertTrue(teller.hasNext());
   assertEquals(teller.next(), 2);
   assertTrue(! teller.hasNext());
   Exception e = null;
   try {
         teller.next();
   } catch (NoSuchElementException nsee) {
         e = nsee;
   }
   assertTrue(e != null);
}
```

c) Forklar hvordan en kan teste at en metode, f.eks. Teller sin next-metode fra deloppgave 2d), kaster et unntak i riktig tilfelle og av riktig type.

En må fremprovosere og håndtere unntaket og i catch-delen/evt. etter next-kallet endre en variabel slik at en etterpå kan teste om unntaket kom/ikke kom (se over). Denne er litt vanskelig, men ble løst i løsningsforslaget fra den ordinære eksamenen.

OPPGAVE 5 (15%): Observatør-observert-teknikken

Du ønsker å gjøre Nummerliste-klassen fra oppgave 3 *observerbar*, i den forstand at den kan si fra til en eller flere *observatører* at et eksisterende nummer er endret eller at et nytt er lagt til. Gitt følgende lyttergrensesnitt:

```
public interface Nummerlistelytter {
   // Kall til  listeEndret-metoden angir at endretListe er endret
   // i posisjon indeks.
   public void listeEndret(Nummerliste endretListe, int indeks);
}
```

a) Anta at en Nummerliste kan ha én eller flere Nummerlistelyttere knyttet til seg. Hvordan må du endre Nummerliste for at den skal fungere som observert og si fra om endringene?

En må 1) innføre attributter og metoder for å håndtere lytterne og 2) sørge for at lyttermetoden blir kalt på rett tidspunkt. En trenger et List-attributt og minst en add-metode for lyttere. Det er naturlig å ha en egen fire-metode med et indeks-parameter, som løper gjennom lytterlista og kaller listeEndret-metoden. Alle metodene som endrer numre-lista må kalle fire med riktig indeks-verdi.

```
private List lyttere;

public Nummerliste() {
   …
   lyttere = new ArrayList();
}

public void setNummer(int indeks, String nummer) {
   ...
   fireListeEndret(indeks);
}

public void addNummer(String nummer) {
   ...
   fireListeEndret(numre.size() - 1);
}

public void addNummerlistelytter(Nummerlistelytter lytter) {
   lyttere.add(lytter);
}

protected void fireListeEndret(int indeks) {
   for (int i = 0; i < lyttere.size(); i++) {
         ((Nummerlistelytter)lyttere.get(i)).listeEndret(this, indeks);
   }
}
```

b) Tegn sekvensdiagram som viser hvordan en observatør legger seg på som lytter, kaller en metode på et Nummerliste-objekt og får beskjed om at listen er endret.

Sekvensen blir som følger:

- Nummerlistelytter-objektet kaller addNummerlistelytter på Nummerliste-objektet.
- Nummerlistelytter-objektet kaller addNummer på Nummerliste-objektet.
- Nummerliste-objektet kaller listeEndret på observatøren.

Det trekkes ikke mye for uryddig notasjon, men diagrammet må inneholde (kun) de to objektene med riktig angitte klassenavn, og navnet på metode(kalle)ne må være riktig angitt.

c) Forklar hvordan du kan bruke en JUnit-testklasse til å teste at Nummerliste implementerer rollen som observert på riktig måte, dvs. kaller listeEndret-metoden på observatørene sine på riktig tidspunkt og med riktige parametre.

Denne er litt vanskelig, det må innrømmes. Problemstillingen ligner litt på testing av om unntak kastes, ved at en må kunne avgjøre om en kodesnutt er utført eller ikke, i dette tilfellet om listeEndret-metoden. Det enkleste er å la testklassen implementere Nummerlistelytter-grensesnittet og registrere denne som lytter til et Nummerliste-objekt. Deretter kaller en metoder på Nummerliste-objektet som er ment å kalle listeEndret, og så sjekkes det om listeEndret faktisk er kalt. Det siste gjøres ved at listeEndret endrer på attributter i testklassen basert på sine parametre, slik at en kan sjekke om den er kalt eller ikke og med riktige parametre.

```
public class Testklasse extends TestCase implements Nummerlistelytter {
   …

   private int forventetIndeks = -1, faktiskIndeks = -1;
   private Nummerliste forventetNummerliste = null, faktiskNummerliste;

   public void listeEndret(Nummerliste endretListe, int indeks) {
      faktiskNummerliste = endretListe;
      faktiskIndeks = indeks;
   }

   public void testNummerlistelytter() {
      Nummerliste liste = new Nummerliste();
      liste.addNummerlistelytter(this);
      forventetNummerliste = liste;
      forventetIndeks = 0; faktiskIndeks = -1;
      liste.addNummer("123");
      assertEquals(faktiskNummerliste, forventetNummerliste);
      assertEquals(faktiskIndeks, forventetIndeks);
      forventetIndeks = 1; faktiskIndeks = -1;
      liste.addNummer("234");
      assertEquals(faktiskIndeks, forventetIndeks);
      forventetIndeks = 0; faktiskIndeks = -1;
      liste.setNummer(0, "345");
      assertEquals(faktiskIndeks, forventetIndeks);
   }
}
```